University of Arizona

CSc 340

Foundations of Computer Systems

Christian Collberg

January 23, 2001

Assembly Code

Machine code vs. Assembly Code

- Machine code is a painful way to program a computer.
- Assembly code is a textual representation of machine code. Mnemonics are used for operation and operand names. Symbolic labels can be assigned to addresses. A program called an assembler translates assembly code into machine code.
- Assembly code, like machine code, varies from CPU to CPU. MIPS assembly code, for example, cannot (easily) be translated into x86 machine code. E.g., MIPS has more registers than x86.

MIPS Assembly Code

- MIPS assembly code consists of four columns: an optional label, an assembler directive or machine operation, the operands, and an optional comment:

```
Label: <tab> OP <tab> arg1,arg2,arg3 <tab> #comment
```

- A label is a symbolic name for an address – it refers to the memory address of the associated instruction or directive. MIPS labels are followed by a colon ⌜:⌝, e.g. ⌜`foo:`⌝.

MIPS Assembly Code

- A directive is a command to the assembler itself. It is not a mnemonic for a machine operation. It tells the assembler to reserve memory, assign constants, etc. Directives start with a period ⌜.⌝, e.g. ⌜`.align 2`⌝:

```
<tab> .align <tab> 2
```

Slide 5–4

MIPS Instruction Format

- A MIPS instruction may have up to three operands (because it's a three-address machine). The names `$r0`-`$r31` can be used to access the registers. Remember that `$r0` is always zero.
- Registers also have alternate names to help you use them according to the convention governing their use. See Table 2 in the SPIM handout. For now remember that `$t0`-`$t9` are for holding temporary values, e.g. a procedure's variables. The contents of these registers may be lost if you invoke a subroutine, however. Use `$s0`-`$s7` if you want the contents preserved.

Slide 5–5

Comments

- Comments start with the '#' character. Everything to the right of a '#' is considered a comment and ignored by the assembler.
- Comments are very important! Assembly code is easier to read than machine code, but still very confusing.
- In most assembly programs **every** line of code contains a comment!

Slide 5–6

Sample MIPS Assembly Code

```
            .data
limit:      .word   10      # number of times to loop
sum:        .word   0       # summation
            .text
## Procedure: main
## Description:
## Sum 1 to limit and store it in sum.
## Registers:
#         $s0:        limit
#         $s1:        sum
#         $s2:        loop counter
```

Slide 5–7

```
main:
      lw        $s0, limit          # $s0 = limit
      lw        $s1, sum            # $s1 = sum
      li        $s2, 0              # $s2 = 0
loop:
      bgt       $s2, $s0, done      # while ($s2<=$s0)
      add       $s1, $s1, $s2       # $s1 = $s1 + $s2
      addi      $s2, $s2, 1         # $s2 = $s2 + 1
      b         loop
done:
      sw        $s1, sum            # sum = $s1
      li        $v0, 10
      syscall                       # exit
```

Slide 5–8

Explanations ...

⌜`.data`⌝ An assembler directive that tells the assembler that what follows is not assembly instructions, but assembler directives relating to memory (data).

⌜`.word`⌝ allocates two words (4-bytes each) of memory. The first word is initialized to 10, and the second to 0. Each of these directives has a label. When a label is used later on in the program, the assembler will replace it with the address for the associated memory location.

Slide 5–9

Explanations ...

⌜`.text`⌝ A directive that tells the assembler that what follows is instructions. The name "text" is historical. These days we might call it "code" or "instructions", but we're stuck with "text".

⌜`lw`⌝ An instruction that loads a value from memory at the address in its second operand into the register specified by its first.

⌜`sw`⌝ An instruction that stores a value from a register into memory.

Slide 5–10

Explanations ...

⌜`bgt`⌝ An instruction that branches to the specified label only if the first operand is greater than the second.

⌜`b`⌝ An instruction that always branches.

⌜`add`⌝ An instruction that adds the second and third operands and stores it in the first. All operands must be registers.

⌜`addi`⌝ An instruction that adds the second and third operands and stores it in the first. The second operand must be an **immediate** value, a 16-bit integer.

Slide 5–11

More (detailed) Explanations...

- The immediate field load and store instructions is only 16 bits wide, so this program assumes that addresses will fit in 16 bits. If this is not the case, you must first load the address into a register, then use the register to load the memory contents:

```
la      $s0, limit
lw      $s1, ($s0)
```

- You can avoid the limit variable by using 10 directly:

```
bgt     $s2, 10, done
```

Note that the immediate value must fit in 16 bits.

Slide 5–12

A Smart Assembler!

The MIPS assembler is actually quite smart and provides a couple of features that you will find both useful and sometimes confusing:

1. You can use ⌜`add`⌝ instead of ⌜`addi`⌝ when adding an immediate to a register. The assembler will replace it with ⌜`addi`⌝.
2. Pseudo-instructions: the assembler supports instructions that the machine does not. When you use one of these instructions the assembler synthesizes it from several machine instructions.

Slide 5–13

A Smart Assembler . . .

3. Instruction reordering: the MIPS computer has delay slots, which means that the values from loads cannot be used immediately, and branch/jump instructions take effect after two cycles. The MIPS assembler reorganizes instructions to deal with these.
4. Because of these when you debug programs you will sometimes see instructions you didn't write, and instructions in a different order than what you wrote.

Slide 5–14

Readings and References

- SPIM Walkthrough handout.
- SPIM S20: A MIPS R2000 Simulator, by James R. Larus. (More recent version on class home page.)
- MIPS Instruction Encoding handout.
- MIPSPro Assembly Language Programmer's Guide from SGI. (On class home page; 129 pages.)